Santander Product Recommendation using ANN, SVD, Collaborative Filtering & Naive Bayes

Team J.A.R.V.I.S - Adrian Chmielewski-Anders, Vincent Yeh, David Friedman, Robert St. Denis, George Burger, Mang Tik Chiu, Jianfu Alvin Situ, Kam Ming Tse, Daljodh Pannu, Jonathan Gee, Andy Tran, Philip Weeks, Joshua Brown, Daniel Ruiz, Niko Solgaard

***Abstract*—We examine the Santander Product Recommendation Challenge from Kaggle [1]. To recommend products, we explored Artificial Neural Networks, Singular Value Decomposition, Collaborative Filtering, and Naive Bayes classificatin methods. Kaggle scores the recommendations of predictions using Mean Average Precision at cutoff 7 (MAP@7). Our results show that the Naive Bayes is the best of the classifiers tested, closely followed by the SVD approach then the Neural Net approach and lastly Collaborative Filtering, suggesting that there is some dependence on the features of the users.**

I. Introduction

Team StatsmanGo also selected the same project, although each team has implemented different methods. Our team's methods are:

1) Artificial Neural Networks
2) Singular Value Decomposition
3) Collaborative Filtering
4) Naive Bayes

The problem is to "predict which products their existing customers will use in the next month based on their past behavior and that of similar customers" [1].

The data provided contains 1.5 years of customer data from Santander Bank. The data starts on January 28th, 2015 and ends May 28th, 2016. It contains monthly records of each customer which contains 24 features and whether or not they are have each of the 24 unique products. This dataset is quite challenging due to its large size ($\tilde{1}$3m rows, 2.1Gb) and the fact that a customer's data can change from month to month (i.e. age, location, product, etc) and also because some people are not customers for all 18 months.

II. Methods

In order to process the data it first needed to be parsed. Training and testing sets were provided by Kaggle and Python 3 was used to parse the data from String and character format into decimal, integer, and binary formats. This resulted in 13 million rows of provided samples with 24 columns of attributes follow by 24 columns of products.

A. *Artificial Neural Networks - ANN*

We constructed an artificial neural network implementing forward propagation and mini-batch gradient descent of size 100 with back-propagation [2]. The input layer contains 24 nodes representing 24 customer attributes; the output layer contains 24 nodes representing 24 products. We iterated through each combination of one to three hidden layers, 24 to 100 hidden nodes, and 0.01 to 1.0 learning rate for 10000 epochs in order to find the most optimized network configuration that produces the least negative log likelihood error.

Network Configuration Parameters	[24 24 24], [24 50 24], [24 100 24], [24 24 24 24], [24 50 50 24]
Learning Rates	0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1

To determine the best configuration for the data, we used 70% of the 13 million samples in the training set to train our network and generate weights. Then, we used the remaining 30% of the training data to evaluate the performance of each configuration and plotted the error of each combination of layers and learning rate. We removed configurations that show no signs of convergence after 10000 epochs from our consideration for the test set. In the testing phase, we chose a couple of the best performing configurations to generate predictions for the testing data.

To simulate prediction of future product puchases, we also separated the samples such that entries with date before 2016-05-28 are used as training set while the remaining samples are used as validation set. We then used the trained model with different hyperparameters to predict which products each customer would have bought in May, 2015. Calculating the MAP@7 scores allowed us to select the configuration that performs best in the testing phase, i.e. which products each customer will purchase in May, 2016.

To test, we parsed the whole testing data in similar fashion as the training data but rather than eliminating rows with NA, we assume the value of NA to be the median of the column. We did this for 200k rows in the testing data (about 22% of the data). Then, we standardized the features using the mean and range from the 13 million samples to ensure the same distribution. We then ran the two differently configured network with the data as input to generate a prediction of likeliness of the product that will be added by the users. Taking the seven most likely products and converted them into their respective product names, along with the user ID, the predictions are outputted into CSV for evaluation on Kaggle, which will then evaluate our result using MAP@7.

B. *Singular Value Decomposition - SVD*

Singular value decomposition is a mathematical decomposition of an $n \times p$ matrix A, into three unique matrices, U, S, and

V. In other words, $A = U \times S \times V^T$ such that:

$$U^T U = I_{n \times n}$$

$$V^T V = I_{p \times p}$$

The singular values S, are the square roots of the eigenvalues from either square, arranged in descending order. Calculating the SVD involves finding the eigenvalues and eigenvectors of AA^T and $A^T A$ [3]. Calculating the full SVD of a very large dataset can be very computationally intensive, however. For example, in our case the dataset contained approximately 13 million samples, which would result in a U matrix of 13 million x 13 million - much too large to be computationally executable using a modest amount of memory.

Thus, in order to use SVD to actually make predictions about the Santander dataset an alternative SVD approach was used such that $A \approx U \times S \times V^T$. To do this we limit the number of eigenvalues returned to some value, k (in our case, the maximum value of k is 48 because there are 48 features in total). This limits the number of vectors being used to reconstruct A to less than are actually needed to fully reconstruct A. Instead, multiplication of U, S and V^T returns A', where each row is still a composite of the singular values, but because there are less rows now have additional noise. This noise is based on the values of all other rows.

The effect of this that for rows with unknown data estimates of purchased products are generated. We improve this process by filling in the unknown purchase values with the mean of the training set. In our case we can treat these values as the likelihood of a the corresponding product being purchased by the user during the next month. In our implementation, we took subsets of the training data provided by Santander, parsed them, and concatenated the vertically with similarly parsed testing data. The testing data was missing the product purchases columns, so these values where filled in with the mean value of these columns in the training data. [4].

C. Collaborative Filtering

Collobarative Filtering is a popular recommender system algorithm based on the assumption that users with similar previous actions will have similar future actions and does not examine any features of users [5]. There exist two methods, namely user-to-user and item-to-item. User-to-user relies on the similarity of users in relation to other users. Item-to-item examines similarities between iterms. User-to-user suffers from scalability problems (in this case we would need to compute $\approx 1,000,000^2$ similarities). Also the distribution of the Santander data is highly skewed because 60.29% of users have a “Current Account” while just 0.0017% of users have the “Guarantees” product. We examined two similarity functions namely Cosine Similarity and Conditional Probability Similarity , they are given by

$$c_1(i,j) = \frac{r_i^T r_j}{\|r_i\|_2 \|r_j\|_2} \tag{1}$$

$$c_2(i,j) = \frac{\text{Freq}(i \cap j)}{\text{Freq}(i)\text{Freq}(j)^\alpha} \tag{2}$$

Where r is the rating column vector and i and j are the $i-$th and $j-$th items, respectively. The hyperparameter α is as a dampening parameter used so that uniquely similar items are more similar than exactly similar items [6]. Note that r_i, r_j in this case will only take values from the set $\{0, 1\}$. Pearsan correlations is not used as it does not work as well emprically [6]. Once a similarity matrix S is obtained where $S_{i,j} = c_x(i,j)$, predictions for a user u and item i can be obtained. There are two preidction functions we looked at, they are

$$p_{u,i}^{(s)} = \frac{\sum_{j \in K} S_{i,j} r_{u,j}}{\sum_{j \in K} |S_{i,j}|} \tag{3}$$

$$p_{u,i}^{(p)} = \sum_{j \in I_u} S_{i,j} \tag{4}$$

Where $r_{u,j}$ is user u's rating for the $j-$th product and K is the set of k items most similar to item i. Note that only previous month of data is considered and that $p_{u,i}^{(p)}$ is called a pseudo-prediction [6].

D. Naive Bayes

”We constructed a Naive Bayes predictor to select the top seven products a customer would want for the next month. Naive Bayes fits classifications to features using conditional probability under the assumption that all features represented are independently distributed. Though this assumption is ’Naive’ it can still often produce decent results. Additionally a Gaussian model was chosen because the features could take on a range of values (e.g age), not only 1 or 0, true or false, which would call for a Bernoulli model. A Gaussian model is preferable to a Multinoulli model because of the wide range of values the features take on. [7] Our predictor could produce Boolean results depending on which class had the higher probability or direct probabilities for ranking the top seven products to recommend.”

III. Results

We give results for MAP@7, given by

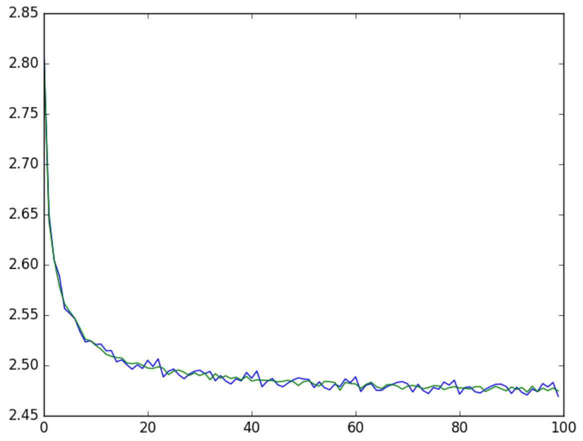
$$q = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{\min{(m,7)}} \sum_{k=1}^{\min{(n,7)}} P(k)$$

where U is the set of all users in two time points m is the actual number of added products for user u and n is the predicted number of products $P(k)$ is the precision at cutoff k. We note that submission to Kaggle only computes this score on roughtly 30% of the data, and thus we have implented this to test with predicting products for the month of May, which we already possess [1]. Note that while the Kaggle score is less accurate because it only uses a subset of the data, our score also suffers because it is based on the previous month for which customers might exhibit a different purchasing propensity. Note that the potential maximum evaluation is 1 if every user added a minimum of 7 products. However, based on the data from the previous months, the trend for the maximum mean average precision is roughly 0.033 since many users do not add any products. This results in a precision of 0 for that user. The maximum for the MAP@7 for May 2016 is 0.0319, so the results from our own implementation of the MAP function is bounded by 0.0319 [8].

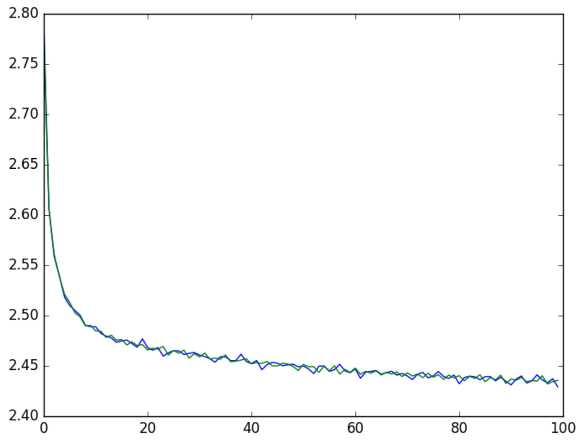
A. Artificial Neural Networks

The training results are summarized in the plots on the next page. The y-axis denotes negative log likelihood and x-axis denotes per 10,000 epoch. Each row of graphs are for a learning rate of [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0], in that order.

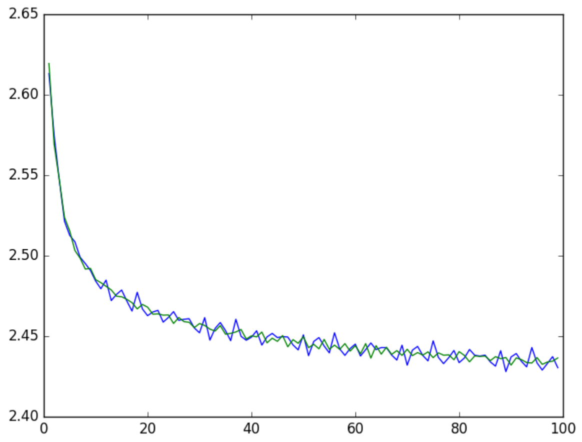
One hidden layer, 24 hidden nodes, $\alpha = 0.5$:



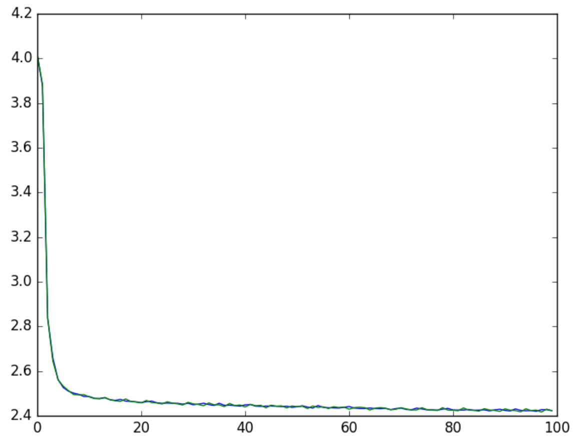
One hidden layer, 50 hidden nodes, $\alpha = 0.5$:



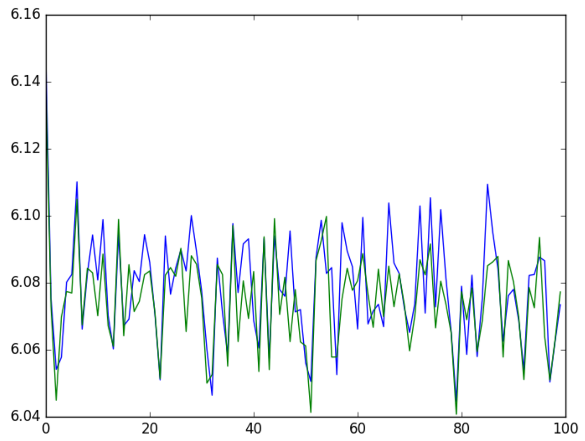
One hidden layer, 100 hidden nodes, $\alpha = 0.5$:



Two hidden layers, 24 hidden nodes per layer, $\alpha = 0.5$:



Two hidden layers, 50 hidden nodes per layer, $\alpha = 0.5$:



After the creation of neural networks, we had to iterate through all combinations of layers, nodes, and learning rates in order to find the most optimal configuration. Using plots with negative log likelihood on the y-axis and per 10000 epochs on the x-axis, we were able to visualize the decline of errors with every iteration. We generated a total of 5 layers/nodes combinations * 7 learning rates = 35 combination plots. Visually examining every plot, we concluded the best learning rate to use was 0.5 because the negative log likelihood fell below 2.45 after the completion of all epochs. We then presented 5 plots representing the 5 layers/nodes combinations ([24 24 24], [24 50 24], [24 100 24], [24 24 24 24], [24 50 50 24]) which we iterated over.

Below is a table of the network configurations that were used to predict the testing set, and their respective Kaggle evaluation scores.

Layers and Nodes	Learning Rates	Evaluation Score
[24 24 24]	0.5	0.0079989
[24 50 24]	0.5	0.0089538
[24 100 24]	0.5	0.0092623
[24 24 24 24]	0.5	0.0088091
[24 50 50 24]	0.5	0.0053822

Using the table above, we concluded that additional hidden layers would lower our evaluation score and the optimal layer to use was 1 hidden layer. Now that we have the optimal learning rate and optimal number of hidden layers, we had to find the optimal number of nodes in our hidden layer to receive the highest evaluation score. Using hidden nodes of 24, 50, 75, 100, 125, we were able to visualize a negative parabolic graph with x-axis as number of nodes and evaluation score on y-axis.

Layers and Nodes	Learning Rates	Evaluation Score
[24 24 24]	0.5	0.0079989
[24 50 24]	0.5	0.0089538
[24 75 24]	0.5	0.0090788
[24 100 24]	0.5	0.0092623
[24 125 24]	0.5	0.0087547

The highest evaluation score reached was at 0.0092623 where the negative parabola reaches its highest point at 100 hidden nodes. Therefore, we concluded the optimal configuration is [24 100 24] with 0.5 learning rate.

B. *Singular Value Decomposition - SVD*

Data Size	k	**MAP@7 30%**	**MAP@7**
1.5m	48	0.0104695	N/A
1.5m	10	0.0085521	N/A
2m	5	0.008444	N/A
3.7m	10	N/A	0.00709095998583

C. *Collaborative Filtering*

We experimented with small (10) and larger (24) values of k and also for α (0.01, 0.1 and 2) the dampening parameter. We have found that for the month of May and 100% of the data for May, the best is the the pseudo-predictor with $\alpha = 0.01$ and the conditional probability function. For Kaggle's 30% of the data for the month of June is $p_{u,i}^{(s)}$ with $k = 10$.

c_x	**Predict**	**k**	α	**MAP@7 30%**	**MAP@7**
c_1	s	10	N/A	0.0050521	0.004378
c_1	s	24	N/A	0.0044793	0.004442
c_2	p	N/A	0.1	0.0057200	0.004723
c_2	p	N/A	0.01	0.0048461	0.004731
c_2	p	N/A	2	0.0036664	0.003034

D. *Naive Bayes*

To test our predictor we held one million samples as test data, roughly the same size as the June test month we were trying to predict for. We trained on the remaining 12 million samples and ran the predictor against the test sample and tested accuracy by counts of true and false, positive and negative predictions. We also measured overall accuracy of predictions as well as accuracy of the products being recommended as that was the overall goal of the predictor. In our final run we trained on all available samples and tested against the test data for June and ranked the product recommendation by the probability of those features having that product. We then filtered out any products already owned by that person in May and created a list of the top seven products for submission. The evaluator ranked our submission at 0.0122565. We also ran the same test with the full sample data before May and predicting on May so we could run the same evaluation and scored 0.0087747. The lower score may be due to our tester running on the full amount instead of only 0.30 or that we had fewer months to use as training data for your predictor.

MAP@7 30%	**MAP@7**
0.0122565	0.0087747

This table shows our predictor's accuracy in guessing the products owned for a particular sample as well as prediction accuracy, total correct positive guesses over total positive guesses made.

IV. Discussion

Our results from the score given by Kaggle show that Naive Bayes performs the best of the results we tested. This is followed closely by the SVD approach and then the Neural Network approach then Collaborative Filtering. Note that in

TABLE I: **Naive Bayes Predictor Accuracy and Correct Recommendation Rate**

	Accuracy	**Recommendation Rate**
Product 1	0.6949	0.0003
Product 2	0.7594	0.0001
Product 3	0.6507	0.6654
Product 4	0.5639	0.0008
Product 5	0.8451	0.1800
Product 6	0.9906	0.0000
Product 7	0.9903	0.0000
Product 8	0.8058	0.3875
Product 9	0.7761	0.0783
Product 10	0.9964	0.0064
Product 11	0.9984	0.0000
Product 12	0.8365	0.0850
Product 13	0.8599	0.1724
Product 14	0.8574	0.0648
Product 15	0.9397	0.0275
Product 16	0.9263	0.0472
Product 17	0.9975	0.0000
Product 18	0.8987	0.1603
Product 19	0.8609	0.1304
Product 20	0.8833	0.1009
Product 21	0.5942	0.0092
Product 22	0.8939	0.1429
Product 23	0.8814	0.1575
Product 24	0.8164	0.2622

general results from the 100% MAP@7 were lower than the scores on Kaggle. This leads one to believe that the month of May there are fewere users with added products compared with June. This shows that the 30% Kaggle score might not be truly indicitave of the real MAP.

A. Naive Bayes

We speculate that Naive Bayes performed the best because the model allowed us to use all of the data in order to compute the probability of features, which is most likely quite accurate given the size of the data. Despite some features being most certainly dependent (for example age and junior account) many features are indeed independ (for example age and current account). Since Naive Bayes assumes all features are independent this turns out to be quite a good predictor. In cases where features are related Naive Bayes can still perform well in spite of dependence due to the dependencies either distributing equally between the two classes or interfering and canceling each other out. [9] (edited)

B. Singular Value Decomposition - SVD

Second was the SVD approach. This approach performed admirably because the SVD removes a lot of unnecessary "noise" in the data and leaves us instead with inferences based on linear combinations of other rows. Given that other rows correspond to other users purchases, these values are often good indicators of future purchases. Additionally by using SVDs, a modification of SVD that uses smaller decomposition matrices, the majority of the training data could be used all at once without extensive online training.

C. Artificial Neural Networks

Artificial Neural Network performed mediocre. It is very likely that our ANN fell into a local minima that was suboptimal which resulted in the lack of accuracy. The set back of using Neural Network was its immense training time that made iterating over every combination of layer, nodes, and learning rate impossible. The best we could do was to make educated guesses and assumptions. Therefore, without high computational technology, it may be hard to achieve an optimal solution using such trial and error method on network topology [10].

D. Collaborative Filtering

Collaborative Filtering was the least effective of the methods tested. This comes from a variety of reasons, perhaps most notably that by design the model does not incorporate any user features. Additionally, many users held only a few products. This made the probability of choosing a new product close to 0. Furthermore, the data for products only indicates whether or not the user holds a particular product at a particular time and does not include the user's rating or opinions on given products. For example a user might not like a product but still have it or like it but this is not give any other increased weight in the prediction.

V. Conclusion

We used several methods to create predictions for the Santander data. We then compared the results of these methods to determine which was the most effective. With the methods we utilized, the Naive Bayes predictor was the most effective based on our data. Based on the scores of our group members, and those of other Kaggle submissions for this topic, there likely exist several ways in which the

models we used in each of our methods could be improved. Many of our methods ignore some of the data features, which may result in the method ignoring potentially useful trends that may lead to better results.
In spite of this, each of our methods managed to come up with a useful prediction. This show us that, at the very least, there are useful predictions to be made about the data, even when considering a subset of the data- image what could be done with a more complex model!

VI. Author Contributions

A. Collaborative Filtering

Adrian Chmielewski-Anders: Collaborative Filtering Code, writeup.
Daljodh Pannu: MAP function, write up.
George Burger: Visualizations for powerpoint, baseline predictor for Collaborative Filtering, write up
Jonathan Gee: Write up, presentation.

B. Neural Networks

Mang Tik Chiu: ANN code, writeup, graphs.
Kam Ming Tse: Writeup, ANN training, helped with implementation.
Vincent Yeh: ANN Writeup, training ANN, helped with implementation.
Jianfu Alvin Situ: Writeup, training ANN, helped with implementation.

C. Singular Value Decomposition

Robert St. Denis: SVD implementation, writeup
David Friedman: SVD implementation, writeup, SVD training.
Andy Tran: Nothing

D. Naive Bayes

Philip Weeks: Naive Bayes implementation, writeup, training, charts.
Daniel Ruiz: Naive Bayes Implementation, writeup.
Niko Solgaard: Parsing, Naive Bayes Implementation.
Joshua Brown: Naive Bayes Implementation

References

[1] S. Bank, "Santander product recommendation," *Kaggle*, 2016. [Online]. Available: https://www.kaggle.com/c/santander-product-recommendation

[2] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[3] [Online]. Available: http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm

[4] S. Gong, H. Ye, and Y. Dai, "Combining singular value decomposition and item-based recommender in collaborative filtering," in *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on*, Jan 2009, pp. 769–772.

[5] V. S. Prem Melville, "Recommender systems." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.3573&rep=rep1&type=pdf

[6] J. T. R. Michael D. Ekstrand and J. A. Konstan, *Collaborative Filtering Recommender Systems*, 2011, vol. 4, pp. 88–97.

[7] J. Brownlee, "Naive bayes for machine learning," Available at http://machinelearningmastery.com/naive-bayes-for-machine-learning/ (2016/05/11).

[8] S. Rajkumar, "Maximum possible score," 2016. [Online]. Available: https://www.kaggle.com/sudalairajkumar/santander-product-recommendation/maximum-possible-score/notebook

[9] H. Zhang, "The optimality of naive bayes," Fredericton, New Brunswick, Canada, Tech. Rep., 2004.

[10] N. Fraser, "Training neural network," *Carleton University*, 1998. [Online]. Available: http://vv.carleton.ca/~neil/neural/neuron-d.html